OntoPDF: using a text mining pipeline to generate enriched PDF versions of scientific papers

Zhu, Yi ; Rinaldi, Fabio

Abstract: In this poster we present a recent extension of the OntoGene text mining utilities, which enables the generation of annotated pdf versions of the original articles. While a text-based view (in XML or HTML) can allow a more flexible presentation of the results of a text mining pipeline, for some applications, notably in assisted curation, it might be desirable to present the annotations in the context of the original pdf document.

OntoPDF: using a text mining pipeline to generate enriched pdf versions of scientific papers

Zhu Yi[*]
zhu-y11@mails.tsinghua.edu.cn

Fabio Rinaldi[†]
fabio.rinaldi@uzh.ch

Abstract

In this poster we present a recent extension of the OntoGene text mining utilities, which enables the generation of annotated pdf versions of the original articles. While a text-based view (in XML or HTML) can allow a more flexible presentation of the results of a text mining pipeline, for some applications, notably in assisted curation, it might be desirable to present the annotations in the context of the original pdf document.

1 Introduction

Several text mining systems are capable of producing a richly annotated version of the input documents, and present it in a format which allows easy browsing. Most of these formats however are text-based. Typically the annotations are overlayed on the text of the original document through simple web technologies (e.g. via XML or HTML plus CSS). However very few systems are capable of producing annotated pdf documents – a notable exception is the Utopia system (Attwood et al., 2010).

Dealing with the original pdf document might be highly desirable in some contexts. For example, database curators are accustomed to the layout of specific journals and often rely on it to quickly locate the information that they are looking for.

The OntoGene text mining system (Rinaldi et al., 2008; Rinaldi et al., 2010) is a pipeline of tools that can provide very advanced biomedical text mining results, as it has been shown by partecipation in several community-organized evaluation campaigns. Recently we have implemented some extensions that allow the OntoGene system to produce annotated pdf documents, overlaying the results of the text mining pipeline on top of the original article.

In the present poster paper we briefly describe the challenges that we had to deal with in order to be able to annotate the pdf documents, how we solved them, and the limitations of the current approach.

2 Algorithm Description

The work described in this paper is based on the assumption that it is possible to create an alignment between the plain text version of the original article and the text in the PDF document. This is not always the case, since PDF is a layout-oriented format, which does not impose restrictions on the order of the elements. In theory it would be possible to create a good looking pdf document by listing its textual elements (e.g. paragraph) in a scrambled order, as long as each of them is given the correct coordinates for correct positioning in the final pdf.

There are several libraries that can extract the textual content of a pdf file, but there is no simple solution to recreate the original textual order if the elements in the pdf do not follow the expected linear order. On the other hand, there is no obvious reason for any editing tool to produce a pdf document with a scrambled order of elements. So it can be assumed that in most cases the pdf document will respect the textual order. If that is the case, it is relatively easy to extract from the pdf file a linear text.

If the original pdf layout does not respect the textual order, advanced pdf to text conversion tools, which allow the definition of specific mappings could be used, such as LApdf (Ramakrishnan et al., 2012)).

The algorithm that we implement can be described as composed of three steps:

1. extraction of text from the pdf

[*]Tsinghua University, Department of Automation, Beijing, China

[†]University of Zurich, Institute of Computational Linguistics, Binzmühlestr. 14, 8050 Zürich, Switzerland

2. alignment of the pdf-derived text with the text mining output

3. creation and insertion of overlays in the pdf document

2.1 PDF text extraction and annotation

We experimented with several pdf libraries in the course of our work, in particular Itext, PDFClown and PDFBox. Both PDFClown and PDFBox gave us relatively satisfactory results. PDFClown[1] provides highlight examples that can be used relatively straightforwardly to annotate all occurrences of a given word in the text, and, like all pdf libraries, it allows the extraction of the textual content of the document. Unfortunately the present version does seem to have some limitation in the recognition of unusual characters, perhaps advanced Unicode. This would not be too bad in itself, since we could simply skip over the unrecognized characters and be able anyway to produce some form of alignment with the plain text version of the document. Unfortunately in such cases PDFClow throws up some low-level exception which cannot be easily catched, so that it is impossible to resume processing from the point where the problem originated.

PDFBox[2] provides a simpler and clearer interface to the underlying pdf: it allows the generation of a sequence of characters with their coordinates on the page. As long the layout of the paper follows the textual order, the sequence of characters extracted with PDFBox can be easily alligned with the plain text version of the document, and the coordinates can be later used to modify the pdf document by inserting suitable overlays. For example it is possible to highlight any character (or word) with a colored rectangle, and even add a hyperlink from a word to an URL providing information on that item. This would allow for example to link up selected entities in the pdf document with entries in a reference database.

2.2 Input Alignment

Since the extracted text is only a sequence of characters and it does not contain any space, we place all the output into a single String object, plus an ArrayList object. The ArrayList consists of the characters plus additional information such as the index of this character in the String object, and its coordinates in the paper. Then we establish a mapping from the index of the String to the index of the ArrayList. After these preparations we can start aligning the input file produced by the text mining system with the text extracted from the pdf document.

The algorithm reads the input file produced by the text mining pipeline, in the format shown in the example below:

W38	142	145	–	All
W39	146	152	–	Rights
W40	153	161	–	Reserved
W41	161	162	–	.
W42	163	175	EFFECT	Inactivation
W43	176	179	–	and
W44	180	190	–	Regulation
W45	191	193	–	of
W46	194	197	–	the
W47	198	205	GC	Aerobic
W48	206	207	–	C
W49	208	209	–	4
W50	209	210	–	–
W51	210	223	–	Dicarboxylate
W52	224	233	–	Transport
W53	234	235	–	(
W54	235	239	GENE	dctA
W55	239	240	–	)
W56	241	245	–	Gene
W57	246	248	–	of
W58	249	260	–	Escherichia

There is one token per line, and each token might have additional properties, such as a type (if it is an entity) and a link to an external database. The alignment algorithm should skip over all tokens that do not have a type attribute, and only for those that have a type, act on the original pdf by adding a color-coded highlighting, which depends on the specific type of the entity.

There are two main methods that could be used to create an alignment between the two sequences:

- Apply an edit distance algorithm to align the extracted String with the sequence of tokens provided by the OntoGene pipeline, for example by calculating the Levenshtein distance.

- Compare word by word between the OntoGene output and the extracted String and try to find the most possible matching word in the extracted String. Then use the mapping to find the corresponding characters in the ArrayList.

[1] http://sourceforge.net/projects/clown/
[2] https://pdfbox.apache.org/

Normally there are well more than 20,000 characters in one paper and thus calculating the Levenshtein distance could be very slow (and certainly much more slower than the second method). Moreover, it is not immediately obvious which costs should be set for substitution, deletion and addition of a character as different papers may result in different alignments. Instead, comparing word by word may be a more brute force approach, but it is actually very efficient and, when the input file and extracted file align well, the results are quite good. Therefore we adopt the second method using word comparison.

Below we illustrate the process with pseudo code, where "sub" denotes the substring of extracted text and "article" the whole String of extracted text. "sub" is used to align with the input text file and will eliminate words in itself when finding the aligned ones. When the word is an aligned entity term (to be annotated), which should in our case be always at the first positions of "sub", we could find the index of "sub" in "article", and then find the index of the entity in "article", thus locating the specific sequence of characters in the ArrayList by using the mapping between "article" and the ArrayList, and then annotating them.

```
0    sub=article;
1    while( term read != null )
2       if( term does not need to be annotated )
3          if( sub.index( term.content ) == 0 )
4          /*Term matched, the order of this term
5          in the extracted text is correct.*/
6          sub = sub.substring( term.content.length );
7          else
8           Code to Process the unmatched situation
9          end
10      else
11         if( sub.index( term.content ) == 0 )
12         sub = sub.substring( term.content.length );
13         Code to find the corresponding index of these
14         characters in the ArrayList and  Annotate the term;
15         else
16         Code to Process the unmatched situation
17         end
18      end
19   end
```

We assume that the sequences in the input file and extracted file are mostly aligned, and in most situations every term will match the first characters of the "sub". Besides, there is also a parameter detecting the index of the term in "sub", which is used in the situations where in the input file some of the information such as copyright claims or author details are omitted. We use a parameter sweep and pick out the parameter that gives the best matching performance.

For the situations that a match for a term in the plain text stream is not encountered in the pdf stream, or the position in "sub" is relatively large, which implies false match, the following two solutions are considered:

- **Term Content has only 1 character**
  As we delete all "-" characters when extracting PDF file in order to skip over any occasional hyphenation of words over a newline, and there are some characters in PDF file that cannot be extracted into the String, we check if sub.index(term.content) will be larger or equal than 2 (meaning if this character is not at the very first positions of the "sub", we will discard it), if yes, we will simply skip this character in the input file and check the next term of input, otherwise we would set "sub" starting after that character and delete it in "sub".

- **Term Content has more than 1 character**
  For this situation, we will temporarily skip this term and set a flag to track if the next term will be at the following position of the skipped term in "sub". If yes, it means two consecutive terms from input file are detected together in the PDF stream, then we can assume that the current corresponding position in "sub" could be modified with a large distance. So we will directly delete all the other characters in between in "sub" and set it starting from the latter detected term; if not, we will skip the first term and continue the iteration, reset the flag for the second term and track the next term.

2.3 Creation and Insertion of Textual Overlays

Since the coordinates of each character are specified according to the page, if we want to create and insert the textual overlays, we have to perform it on the exact page, so we need an array recording how many characters there are in each page so that we can easily recover the correct page for an annotation. After finding the page, we can obtain a library object ("annots") which will allow insertion of overlays into that page.

To create the Highlight and Hyperlink overlays for each term entity, we have to set up a markup containing all the information to be inserted in the PDF file. We need to set the color and the type (Highlight or Hyperlink), and a rectangle and a group of

quadpoints both encompassing the areas of this annotation. The rectangle could be determined by the coordinates of 2 diagonal points: the lower left point and the upper right point, and the quadpoints are effectively an array with 8 elements, where each pair denotes the X and Y coordinates of the 4 points representing the rectangle.

We first count the number of the characters in the term, and set the location of the first character to the lower left point of the rectangle, and the location of the last character plus a Y direction translation of the font size distance to the upper right point. The quadpoints can also be determined accordingly. When encountering the situation where the first and the last character of the term are not in the same line, we have to create 2 markups to cover characters in different lines, setting the first rectangle and quadpoints pair ending at the last character that shares the same line with the first one, and the second pair starting from the second line, covering the rest of characters of the term. After finishing the creation of overlays, we can add them into the "annots" object to complete the insertion into the page.

3 Problems

Unfortunately not all pdf files behave as nicely as it is required for this algorithm to function correctly. However it is in general possible to predict if the algorithm will function correctly simply by inspecting the "creator" property of the pdf file, which contains the name of the software tool that was used to produce it. We have been able to produce an initial list of "well behaved" pdf creation tools, for which we can guarantee good performance with our algorithm.

Notwithstanding the satisfactory results, there remains some problems to be considered in our future work:

- The input file parses the PDF file in the order of columns in one page, and the extracted file is in the logical order, so it may cause some logical problems when doing alignment, thus leading to missing annotations. For example, a remark or footnote at the bottom of the first column may appear before the second column in the input file, and in the extracted file it will appear obviously after the second column.
- Some character in the PDF file cannot be recognized by the conversion tool, so there is always some mismatch during the alignment.
- In some case, the input file will contain more words than the extracted file or less. We have managed to distinguish this situation by setting a flag to deal with individual word mismatch. But if there are several redundant words in consecutive positions, the algorithm will sometimes fail to detect them, and cause a misalignment.
- The alignment algorithm is applied both to words that require annotations and words that do not. In situations where it fails to align them, we choose to ignore these mismatches. In some cases this could lead to missing annotations.

Figure 1 shows an example of a document annotated using the OntoPDF approach.

4 Conclusion

We have presented a simple but effective method for enriching existing pdf documents with textual overlays marking the entities detected by a text mining system. We believe that this approach could be useful in some applications of biomedical text minining, notably assisted curation.

References

TK Attwood, DB Kell, P McDermott, J Marsh, SR Pettifer, and D Thorne. 2010. Utopia documents: linking scholarly literature with research data. *Bioinformatics*, 26(18):i568–i574.

Cartic Ramakrishnan, Abhishek Patnia, Eduard Hovy, and Gully APC Burns. 2012. Layout-aware text extraction from full-text pdf of scientific articles. *Source Code for Biology and Medicine*, 7(7).

Fabio Rinaldi, Thomas Kappeler, Kaarel Kaljurand, Gerold Schneider, Manfred Klenner, Simon Clematide, Michael Hess, Jean-Marc von Allmen, Pierre Parisot, Martin Romacker, and Therese Vachon. 2008. OntoGene in BioCreative II. *Genome Biology*, 9(Suppl 2):S13.

Fabio Rinaldi, Gerold Schneider, Kaarel Kaljurand, Simon Clematide, Therese Vachon, and Martin Romacker. 2010. OntoGene in BioCreative II.5. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(3):472–480.

JOURNAL OF BACTERIOLOGY, Sept. 1999, p. 5624–5635 Vol. 181, No. 18
0021-9193/99/$04.00+0

Inactivation and Regulation of the Aerobic C_4-Dicarboxylate Transport (*dctA*) Gene of *Escherichia coli*

SUZANNE J. DAVIES,[1] PAUL GOLBY,[2] DAVOOD OMRANI,[1]† SUSAN A. BROAD,[2] VIKKI L. HARRINGTON,[2] JOHN R. GUEST,[1] DAVID J. KELLY,[1] AND SIMON C. ANDREWS[2]*

Krebs Institute for Biomolecular Research, Department of Molecular Biology and Biotechnology, University of Sheffield, Sheffield S10 2TN,[1] and School of Animal & Microbial Sciences, University of Reading, Whiteknights, Reading RG6 6AJ,[2] United Kingdom

Received 16 February 1999/Accepted 2 July 1999

The gene (*dctA*) encoding the aerobic C_4-dicarboxylate transporter (DctA) of *Escherichia coli* was previously mapped to the 79-min region of the linkage map. The nucleotide sequence of this region reveals two candidates for the *dctA* gene: *f428* at 79.3 min and the *o157a-o424-o328* (or *orfQMP*) operon at 79.9 min. The *f428* gene encodes a homologue of the *Sinorhizobium meliloti* and *Rhizobium leguminosarum* H^+/C_4-dicarboxylate symporter, DctA, whereas the *orfQMP* operon encodes homologues of the aerobic periplasmic-binding protein-dependent C_4-dicarboxylate transport system (DctQ, DctM, and DctP) of *Rhodobacter capsulatus*. To determine which, if either, of these loci specify the *E. coli* DctA system, the chromosomal *f428* and *orfM* genes were inactivated by inserting Sp^r or Ap^r cassettes, respectively. The resulting *f428* mutant was unable to grow aerobically with fumarate or malate as the sole carbon source and grew poorly with succinate. Furthermore, fumarate uptake was abolished in the *f428* mutant and succinate transport was ~10-fold lower than that of the wild type. The growth and fumarate transport deficiencies of the *f428* mutant were complemented by transformation with an *f428*-containing plasmid. No growth defect was found for the *orfM* mutant. In combination, the above findings confirm that *f428* corresponds to the *dctA* gene and indicate that the *orfQMP* products play no role in C_4-dicarboxylate transport. Regulation studies with a *dctA-lacZ* (*f428-lacZ*) transcriptional fusion showed that *dctA* is subject to cyclic AMP receptor protein (CRP)-dependent catabolite repression and ArcA-mediated anaerobic repression and is weakly induced by the DcuS-DcuR system in response to C_4-dicarboxylates and citrate. Interestingly, in a *dctA* mutant, expression of *dctA* is constitutive with respect to C_4-dicarboxylate induction, suggesting that DctA regulates its own synthesis. Northern blot analysis revealed a single, monocistronic *dctA* transcript and confirmed that *dctA* is subject to regulation by catabolite repression and CRP. Reverse transcriptase-mediated primer extension indicated a single transcriptional start site centered 81 bp downstream of a strongly predicted CRP-binding site.

Escherichia coli can utilize C_4-dicarboxylates as a carbon and energy source under aerobic and anaerobic conditions (9, 50, 56). Anaerobically, the uptake, exchange, and efflux of C_4-dicarboxylates (fumarate, malate, maleate, and succinate) and L-aspartate are mediated by the three independent dicarboxylate uptake (Dcu) systems, DcuA, DcuB, and DcuC (9, 12, 13, 50, 56). These Dcu systems appear to be active solely under anaerobic conditions (9).

Aerobically, uptake of C_4-dicarboxylates is mediated by a secondary transporter and/or a binding-protein-dependent system, designated Dct (20, 24). The Dct system has an apparent K_m of 10 to 20 μM for C_4-dicarboxylates and is driven by the electrochemical proton gradient (15), and its activity is induced by succinate and is subject to catabolite repression (20, 27). The corresponding *dctA* mutants cannot utilize the C_4-dicarboxylates malate and fumarate but grow normally on the monocarboxylate lactate (27). Transport across the outer membrane may be mediated by a C_4-dicarboxylate-binding protein (Cbt; K_d for C_4-dicarboxylates of 30 to 50 μM) and a porin (3, 4, 25–30).

Three genetic loci (*cbt* at 16.6 min, *dctA* at 79.3 min, and *dctB* at 16.4 min) are involved in aerobic C_4-dicarboxylate transport (27). The nucleotide sequence of the 76- to 81.5-min region revealed a putative *dctA* gene (*f428*) encoding a protein (DctA) most closely resembling the DctA proteins of *Sinorhizobium meliloti* and *Rhizobium leguminosarum* (62 to 63% identity) that function as H^+/C_4-dicarboxylate symporters (51). The DctA proteins are members of a family that includes the Na^+/H^+ glutamate symporters (GltP/GltT). A role for the putative *dctA* gene of *E. coli* in the utilization of C_4-dicarboxylates (and the cyclic monocarboxylate orotate) has been suggested by complementation studies with *Salmonella typhimurium* *dctA* or *outA* mutants (2, 51). The coding regions corresponding to the *dctB* (predicted to encode an inner membrane protein) and *cbt* (predicted to encode the binding protein) genes have yet to be identified (23).

In addition to the *dctA* (*f428*) gene, *E. coli* contains three apparently cotranscribed genes (*o157-o424-o328* or *orfQMP*) at 79.9 min that encode products 23 to 32% identical to the DctPQM components of the periplasmic binding protein-dependent C_4-dicarboxylate transport system of *Rhodobacter capsulatus* (11, 46, 51). The *orfQMP* genes are apparently part of a large operon involved in pentose sugar metabolism (11, 42). This suggests that the *orfQMP* products form a pentose sugar transporter, although, given their similarity to the *R. capsulatus* DctPQM components, it is also possible that they transport C_4-dicarboxylates.

* Corresponding author. Mailing address: School of Animal & Microbial Sciences, University of Reading, Whiteknights, P.O. Box 228, Reading RG6 6AJ, United Kingdom. Phone: 118-987-5123, ext. 7045/7886. Fax: 118-931-0180. E-mail: s.c.andrews@reading.ac.uk.

† Present address: Department of Genetics, Uromeiya Medical School, Uromeiya, Iran.

Figure 1: Example of annotated PDF document